Probabilistic Performance Modelling when using Partial Reconfiguration to Accelerate Streaming Applications with Non-Deterministic Task Scheduling

Bruno da Silva[1,2][0000−0002−4877−9688], An Braeken[2][0000−0002−9965−915X], and Abdellah Touhafi[1,2][0000−0001−8891−180X]

[1] Vrije Universiteit Brussel (VUB), ETRO Department, Brussels, Belgium
[2] Vrije Universiteit Brussel (VUB), INDI Department, Brussels, Belgium
{bruno.da.silva,an.braeken,abdellah.touhafi}@vub.be

Abstract. Many streaming applications composed of multiple tasks self-adapt their tasks' execution at runtime as response to the processed data. This type of application promises a better solution to context switches at the cost of a non-deterministic task scheduling. Partial reconfiguration is a unique feature of FPGAs that not only offers a higher resource reuse but also performance improvements when properly applied. In this paper, a probabilistic approach is used to estimate the acceleration of streaming applications with unknown task schedule thanks to the application of partial reconfiguration. This novel approach provides insights in the feasible acceleration when partially reconfiguring regions of the FPGA are partially reconfigured in order to exploit the available resources by processing multiple tasks in parallel. Moreover, the impact of how different strategies or heuristics affect to the final performance is included in this analysis. As a result, not only an estimation of the achievable acceleration is obtained, but also a guide at the design stage when searching for the highest performance.

1 Introduction

Streaming applications are present in a wide range of domains such as digital signal processing, audio, and imaging, which require several compatible modes or configurations only active based on pre-defined contexts. Such dynamic streaming applications are able to adapt their response as reaction to an environmental change [9], [13], [2], [5]. For instance, multifunction array radars based on a phased array need to execute multiple integrated functions such as tracking, surveillance, communication, calibration or counter measures in an unspecific order [12]. The multifunction radar has to search in multiple regions, which are sub-divided into beam positions with each position executing a task based on the previous monitoring operation. Hence, it is not possible to determine in advance what operations or tasks need to be computed at a certain moment [9]. However, such type of streaming applications can achieve high performance on

Table 1. *Example of a cost table.*

Task (T_i)	Probability (p_i)	Time Cost (t_i)	Area Cost (a_i)	Compatibility (RM_i)
T_A	1/3	t_A	1	RM_{T_A}
T_B	1/3	t_B	1/2	RM_{T_B}
T_C	1/3	t_C	1/4	RM_{T_C}

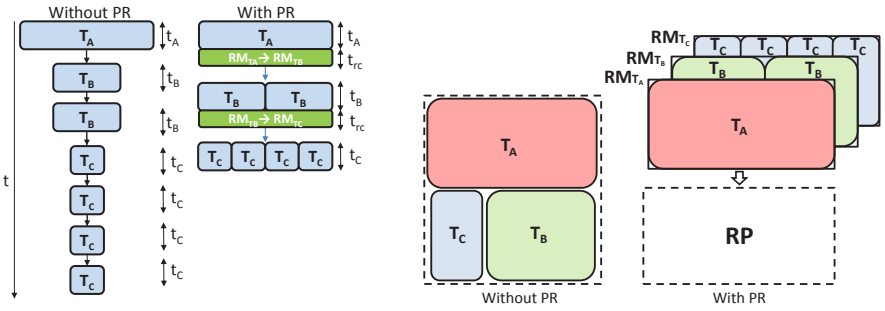

Fig. 1. *Example of how PR can be used to increase performance (left) while maximizing the reuse of the reconfigurable resources (right). The dashed framed area represents the area consumption with and without PR.*

FPGAs when they are properly designed to exploit pipeline-level and instruction-level parallelism [8]. We believe that the use of Partial reconfiguration (PR) with the proper heuristics can further improve the performance. PR is a unique feature of FPGAs which allows the change of the functionality of reconfigurable partitions (RP) on the FPGA at runtime. The use of PR for such type of applications, where the order and the type of the tasks to be executed are not known in advance, might not justify the additional design effort to achieve residual performance acceleration.

In this paper, we not only present a general methodology to increment performance by using PR to maximize the area reuse but also a probabilistic approach to predict the achievable speedup. This probabilistic performance model provides performance insights at the design time, helping to decide parameters like the size of the RP or to evaluate the PR performance overhead. The principles of the methodology to exploit PR to accelerate a streaming application with unknown task scheduling are depicted in Figure 1. The execution without PR requires $t_A + 2 \cdot t_B + 4 \cdot t_C$ units of time to complete one execution, where t_i corresponds to the computation time of a task i detailed in Table 1. By partially reconfiguring one RP with other configurations, called reconfigurable modules (RM), like for instance RM_{T_B} or RM_{T_C}, multiple tasks can be computed in parallel through the exploitation of the unused resources. The overall execution time is then reduced to $t_A + t_B + t_C + 2 \cdot t_{pr}$, where t_{pr} is the time overhead due to reconfiguring the RP. Moreover, this approach also leads to area savings as illustrated in Figure 1. Instead of dedicating area to allocate each type of task, one RP can be properly dimensioned to not only allocate one instance of the

most area demanding task, but also multiple instantiations of low-area demanding tasks (e.g. T_B or T_C in this example). This naive example, however, can become significantly more complex when considering multiple RPs computing hundreds of tasks that can be merged to share resources. As a result, the design effort that is required to fully exploit PR for performance acceleration is not negligible. The allocation of the tasks on the available RPs, the combination of multiple tasks to reuse resources or the schedule of the merged tasks on the reconfigurable RP are challenges that our approach helps to predict. The ultimate goal of our approach is to accurately predict the achievable acceleration when using the proposed methodology to exploit the benefits of PR.

The use of PR to support multiple configurations and to improve performance of streaming applications with unpredictable scheduling has been already used in [4] to accelerate a platform supporting PR through PCIe [3]. Although the approach used in [4] proposes heuristics to increase area reuse and performance, we present in this paper a general methodology to increase performance. For instance, our methodology introduces a parameter to reflect the reconfiguration cost which does not necessarily stands for PCIe-based reconfiguration like in [3], [4] but it is also applicable for other reconfigurations interfaces like the Internal Configuration Access Port (ICAP) on Xilinx FPGAs. Moreover, a probabilistic approach to predict the achievable performance when using PR is here presented in order to reduce the design effort required to apply the proposed methodology. This probabilistic performance modelling can be used to evaluate different strategies or heuristics targeting either area savings or performance improvements without the need of implementing them on the FPGA. The main contributions of this work can be summarized as follows:

- We present a generalized heuristics-based methodology to exploit PR in order to accelerate streaming applications.
- Our approach represents the basis for probabilistic performance predictions when using PR to accelerate streaming applications.

This paper is organized as follows. Section 2 presents related work. The methodology to use PR for performance acceleration is described in Section 3. In Section 4 the problem formulation and the equations to predict performance based on the application's characteristics are introduced. An audio streaming application is used to validate the performance predictions when applying our methodology. The results are presented in Section 5. Finally, our conclusions are drawn in Section 6.

2 Related Work

Different performance prediction models when using PR have been proposed in the last decade. The authors in [6], [7] present a theoretical analysis of the performance bounds of PR. The basis of their analysis is the full decomposition of the application in tasks. The tasks' timings and the operations involved in the PR are used to estimate performance bounds and speedups. Similarly, the

authors in [10] propose a cost model to determine the performance impact of PR. Both approaches, however, are not directly applicable to applications with an unpredictable tasks' scheduling.

Different strategies using PR to maximize the area reuse or to increase performance have been proposed. The authors in [14], [15] present a novel approach for the resource sharing of RPs by merging tasks of streaming applications with an unpredictable tasks' scheduling by identifying similarities between tasks. However, their approach targets the minimization of the FPGA reconfiguration time by optimizing the allocation of the applications on the FPGA rather than incrementing the area reuse per RP. The authors in [1] present solutions to reduce the PR cost. Their approach, consisting of an Integer-Linear Programming (ILP) and a heuristic to exploit PR techniques such as *module reuse*, does not consider the use of PR to increment the resource sharing of the RPs. Our methodology not only addresses similar types of applications, but also reduces the number of reconfigurations while prioritizing the area reuse of RPs by taking advantage of similarities between tasks to share logic resources of RPs. In addition, our probabilistic approach enables performance estimation at the design time, leading to a reduction of the overall design effort.

3 Proposed Methodology

Our generalized methodology exploits PR to accelerate streaming applications with non-deterministic task scheduling. This methodology consists of the three heuristics depicted in Figure 2. An initial classification identifies the type of incoming tasks and tags them based on the Cost Table (CT). The merging heuristic groups tasks to be executed in parallel in the same RP based on their compatibility. The execution of all the tasks is split in iterations or time slots based on the number of tasks and RPs. Each RP has a dedicated task's queue which determines the time slot when the tasks are executed in the RP. However, scheduling strategies are needed in order to minimize the PR impact, leading to the desired performance [9]. A scheduling heuristic distributes the merged tasks between the available time slots of the RPs. As a result, the heuristics allow a performance acceleration by exploiting the area reuse.

Different heuristics can be applied on this methodology. For instance, the authors in [1] propose a scheduling heuristic targeting performance compatible with this methodology. A merging heuristic to exploit the compatibility of similar tasks, leading to a higher reuse of the available area by allocating different types of tasks in the same RM has been proposed in [4]. In any case, the proposed methodology increases the area reuse by computing compatible tasks in the same RP to accelerate the overall performance. Moreover, the proposed probabilistic approach can be adjusted to reflect the characteristics of the heuristics, like done in Section 4 for a merging and scheduling heuristics.

The usability of this general methodology is determined by the information available in the CT. The construction of such a table does not demand additional effort than some profilings and measurements at the design stage. The probability

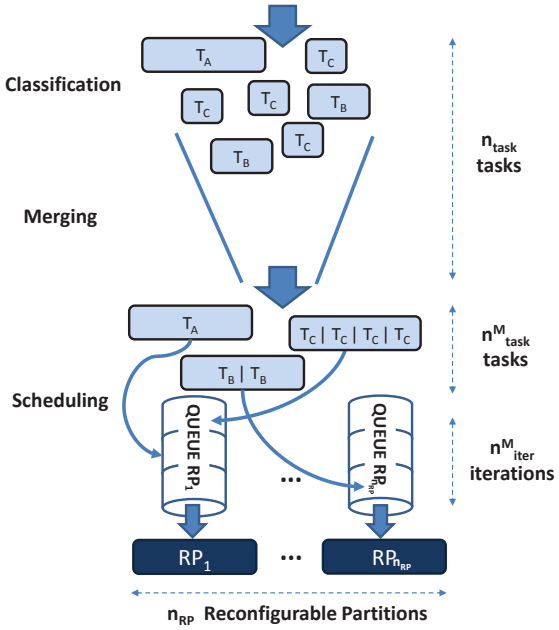

Fig. 2. *Overview of the proposed methodology to exploit PR for performance accelerations.*

of executing a specific task is defined by the application's characteristics and can be estimated in advance or obtained after multiple executions. The area cost of each task (a_i) is already known when implementing the task on the target FPGA. The number of tasks that can be allocated on the RP, which is hereby called the level of parallelism (LP), can be obtained based on the available resources in the RP. The parameter a_i represents the relative area demands of a task i in terms of one RP, which corresponds to the inverse of LP. he time cost (t_i) is obtained through simulations at high level or execution profiles on the FPGA once the task is implemented. The compatibility depends on the level of the desired area reuse, the available I/O and the task's characteristics. The area reuse can be increased when combining the execution of the same, or even different types of tasks on the same RM, like proposed in [4]. The available I/O bandwidth determines, together with the demanded area consumption, the value of LP. The task's characteristics determine if multiple tasks of different types can be executed in parallel. For instance, a high variance of t_i might lead to performance degradation when merging tasks with different t_i, since the merged tasks would have to wait for the most time demanding one. Finally, the size of PR must be adjusted to provide enough resources to allocate the most area demanding task. Nonetheless, its size can be enlarged to be able to allocate multiple instances of other tasks. Despite such enlargements would reduce the area savings, they might increase performance.

The cost of PR is reflected in t_{pr}, which is the time needed to reconfigure one RP. Moreover, t_{pr} increases with the amount of resources available in the RP, and must be considered when determining the size of the RP. The area overhead due to supporting PR is not considered in our approach and is assumed significantly lower than the area demanded by the streaming application's tasks.

The proposed methodology targets streaming applications with the following characteristics:

- *Non-deterministic scheduling:* The tasks and their schedule are not known in advance.
- *Non-priority tasks:* The tasks can be executed without a priority order.
- *Non-data dependencies between tasks:* The tasks in the same execution are not data dependent.

Notice that the non-data dependency between tasks enables the execution of tasks without any priority order. This is not a strong constraint for our approach since it can be overcome by considering data dependencies during scheduling. However, a priority order might reduce the achievable performance acceleration.

Applications like smart cameras adapt their response to the environmental context, demanding runtime decisions under unknown beforehand conditions [13]. The processing of SQL queries can also be accelerated by using PR when treating each type of SQL query as independent task. Each SQL query presents different query plan which can be implemented with a different architecture, like proposed in [11].

4 Problem Formulation

The following probabilistic approach intends to predict the achievable speedup obtained by using the described methodology. Let us consider a streaming application (Figure 2) composed of a certain number of tasks (n_{tasks}) with a a number of different types (n_{types}), which must be executed without following any particular schedule. At a certain instant, a number of independent tasks (n^I_{tasks}) must be scheduled to be executed on the FPGA. The probability of having a task i (T_i) is p_i. Each T_i demands a computational time (t_i) of the FPGA. Finally, t_{pr} is the time cost of the PR of one of the reconfigurable partitions (RPs) of the FPGA, which has a certain number of RPs (n_{RP}) available.

The tasks are grouped to be executed in parallel based on their area and I/O bandwidth demand as part of the design flow. The level of parallelism of each T_i is LP_i. As a result, several RMs compatible with the available RPs are generated to allocate all the tasks of the streaming application. A CT including the time cost of the tasks, their area cost and the compatible RMs is generated to be used by a set of heuristics designed to exploit the area reuse and to optimize the merged tasks' scheduling. The overall acceleration is determined by three properties: the number of available RPs (n_{RPs}), the average LP achieved by merging tasks, and finally, the scheduling of the tasks. The performance impact of the last two characteristics are firstly analysed from a probabilistic point of view.

4.1 Probabilistic Approach

Our methodology (Section 3) assumes that each execution on the FPGA is composed of mutually exclusive and independent n^I_{tasks}. The probability of a T_i in n^I_{tasks} follows a multinomial distribution. However, for a particular task it can be approximated to a binomial. Thus, the probability of having r tasks T_i in n^I_{tasks} is:

$$\begin{aligned} P(T_i = r) &= \binom{n^I_{task}}{r} \cdot {p_i}^r \cdot (1-p_i)^{(n^I_{tasks}-r)} \\ &= \frac{n^I_{task}!}{r!\left(n^I_{tasks}-r\right)!} \cdot {p_i}^r \cdot (1-p_i)^{(n^I_{tasks}-r)} \end{aligned} \tag{1}$$

The average execution time needed (t_{exec}) is:

$$t^I_{exec} \;=\; \left\lceil \frac{n^I_{tasks}}{n_{RPs}} \right\rceil \cdot \sum^{n_{types}} p_i \cdot t_i \tag{2}$$

which is simplified to Eq. 3 when assuming only one RP:

$$t^I_{exec} \;=\; n^I_{tasks} \cdot \sum_{i}^{n_{types}} p_i \cdot t_i \tag{3}$$

The average area cost (A_{cost}) is defined based on the task's relative area cost (a_i) and their probability (p_i):

$$A_{cost} \;=\; \sum_{i}^{n_{types}} p_i \cdot a_i \tag{4}$$

4.2 Merging

As a consequence of the tasks' merging, n^I_{tasks} is reduced, the n^I_{types} becomes dependent of the number of RMs (n_{RMs}) and p_i is modified. The parameters involved in the tasks' merging have the following conditions:

$$n^M_{types} \leq n_{RMs} \tag{5}$$

$$n^M_{tasks} \leq n^I_{tasks} \leq n_{tasks} \tag{6}$$

where n^M_{types} is the different types of merged tasks and n^M_{tasks} is the number of merged tasks to be computed. Finally, p^M_i is the probability of having one particular type of merged task i. The value of these parameters depends on the CT and the type of the tasks' merging supported.

The example CT depicted in Table 1 shows how some of the tasks can be allocated in the same RP. The compatibility list reflects that only the same type of tasks can be merged, since an unique RM is exclusively dedicated to compute

each type of task. As a result of the tasks' merging, the initial parameters of our approach are modified for a post-merging analysis. Hence, the probability p_i^M after merging tasks T_i becomes:

$$p_i^M = p_i \cdot \frac{a_i}{\sum_j^{n_{types}^I} a_j} \cdot \left(\sum_k^{n_{types}^M} \frac{a_k}{\sum_m^{n_{types}^I} a_m} \right)^{-1} = p_i \cdot a_i^M \tag{7}$$

where a_i^M represents the demanded area for the merged type of tasks i.

The computation time of T_i after merging tasks (t_i) is not modified by computing LP_i tasks in parallel (Eq 8).

$$t_i^M = max(t_i) = t_i \tag{8}$$

Notice that t_i^M would be the maximum of t_i when merging different types of tasks.

The number of different types of tasks after merging (n_{types}^M) depends on the n_{types}^I (Table 1) and the probability of having a task of each type. This is only true when merging the same type of tasks. If two compatible types of tasks are merged, sharing the same RM, the value of n_{types}^M will be lower than n_{types}^I. The approach can be adjusted to merge compatible types of tasks by accumulating their probability. For the sake of simplicity, the following analysis only considers the merging of the same type of tasks.

$$n_{types}^M = n_{types}^I \cdot \sum_i^{n_{types}} p_i = n_{types}^I \tag{9}$$

Notice that n_{types}^I does not need to be equal to n_{types}. In fact, n_{types}^I is obtained by considering the tasks' probabilities and n_{task}^I since it follows a multinomial distribution. Finally, the number of tasks after merging (n_{tasks}^M) depends on LP and the probability of having a certain type of task (Eq 10).

$$n_{tasks}^M = n_{tasks}^I \sum_i^{n_{types}^I} p_i \cdot a_i \tag{10}$$

The execution time after merging tasks, based on Eq (10), is similarly defined like Eq (3):

$$t_{exec}^M = \left(\sum_i^{n_{types}^I} p_i \cdot a_i \right) \cdot n_{tasks}^I \cdot \sum_i^{n_{types}^I} p_i \cdot t_i = \left(\sum_i^{n_{types}^I} p_i \cdot a_i \right) \cdot t_{exec}^I \tag{11}$$

where $\sum_i^{n_{types}^I} p_i \cdot a_i$ must be lower than 1 in order to have acceleration. Therefore, the theoretical acceleration by merging tasks ($Speedup_{th}$) is defined as:

$$Speedup_{th} = \frac{t_{exec}^I}{t_{exec}^M} = \frac{t_{exec}^I}{t_{exec}^I \cdot \left(\sum_i^{n_{types}^I} p_i \cdot a_i \right)} = \frac{1}{\sum_i^{n_{types}^I} p_i \cdot a_i} \tag{12}$$

Notice that the PR cost is not included yet in the theoretical acceleration. In fact, this acceleration is reduced when considering the PR cost. Let p_{pr} be the probability of PR and t_{pr} the time cost of such partial reconfiguration. Eq (11) is readjusted as follows:

$$t^M_{exec} = \left(\sum_i^{n^I_{types}} p_i \cdot a_i \right) \cdot \left(t^I_{exec} + n^I_{tasks} \cdot t_{pr} \cdot p_{pr} \right) \tag{13}$$

Therefore, the achievable acceleration thanks to merging tasks ($Speedup$) becomes:

$$\begin{aligned} Speedup &= \frac{t^I_{exec}}{\left(\sum_i^{n^I_{types}} p_i \cdot a_i \right) \cdot \left(t^I_{exec} + n^I_{tasks} \cdot t_{pr} \cdot p_{pr} \right)} \\ &= Speedup_{th} \cdot \frac{t^I_{exec}}{t^I_{exec} + n^I_{tasks} \cdot t_{pr} \cdot p_{pr}} \\ &= Speedup_{th} \cdot PR_{cost} \end{aligned} \tag{14}$$

where PR_{cost} represents the performance degradation due to PR and ranges from 0 to 1.

$$PR_{cost} = \frac{t^I_{exec}}{t^I_{exec} + n^I_{tasks} \cdot t_{pr} \cdot p_{pr}} \tag{15}$$

and, by applying Eq. 3, can be simplified to

$$PR_{cost} = \frac{\sum_i^{n_{types}} p_i \cdot t_i}{\sum_i^{n_{types}} p_i \cdot t_i + t_{pr} \cdot p_{pr}} \tag{16}$$

Due to the fact that PR_{cost} might be lower than 1, there is acceleration only if:

$$Speedup_{th} > \frac{1}{PR_{cost}} \tag{17}$$

4.3 Scheduling

A proper tasks' scheduling minimizes the impact of PR by reducing the number of PR (n_{pr}). The value of n_{pr} is directly related to p_{pr} and the merged n^M_{tasks}. Moreover, p_{pr} is determined by the supported RMs and the configuration of the RP at a certain instant.

No Scheduling The number of iterations in one execution after merging (n^M_{iter}) is expressed as

$$n^M_{iter} = \left\lceil \frac{n^M_{tasks}}{n_{RP}} \right\rceil \tag{18}$$

which equals n^M_{tasks} when considering only one RP. Hereby, only one RP is assumed ($n_{RP} = 1$) for the sake of simplicity while introducing the probabilistic approach.

Let us consider $i \in < 1, ..., n_{RPs} >$ and $j \in < 1, ..., n_{RMs} >$. The probability to reconfigure a RP_i configured with a RM_j at a certain iteration $kj \in < 1, ..., n^M_{iter} >$ can be expressed as:

$$\begin{aligned} p_{pr} &= P(RP_i[k] \neq RP_i[k-1]) \\ &= P(RP_i[k] = RM_j \cap RP_i[k-1] \neq RM_j) \end{aligned} \tag{19}$$

Notice that $RP_i[0]$ represents the initial configuration of the RP_i. Each iteration can be considered independent when there is no tasks' scheduling. Hence, p_{pr} can be expressed as:

$$p_{pr} = P(RP_i[k] = RM_j) \cdot P(RP_i[k-1] \neq RM_j) \tag{20}$$

which, based on Eq. 7, is reduced to:

$$p_{pr} \;=\; \sum_{idx=1}^{n^M_{types}} p^M_{idx} \cdot (1 - p^M_{idx}) \tag{21}$$

Without any tasks' scheduling the probability p_{pr} equally affects to each RP. Similarly, p_{pr} is independent between iterations. Hence, n_{pr} is expressed based on Eq. 18 as:

$$n_{pr} \;=\; n^M_{tasks} \cdot p_{pr} \tag{22}$$

Therefore, Eq. 16 can be expressed as:

$$PR_{cost} \;=\; \frac{t^I_{exec}}{t^I_{exec} + t_{pr} \cdot n_{pr}} \tag{23}$$

Iteration-oriented Scheduling The iteration-oriented scheduling heuristic proposed in [4] exploits the previous configuration of the available RPs to reduce n_{pr}. This strategy searches for those tasks in the available n^M_{tasks} compatible with the configuration of a RP at a certain iteration. The probability of having at least one task i in n^M_{tasks} is equivalent to

$$P(n_i > 0) = 1 - P(n_i = 0) \tag{24}$$

where n_i is the number of tasks i in n^M_{tasks}. This probability is calculated as a binomial distribution:

$$P(n_i = 0) = (1 - p^M_i)^{n^M_{tasks}} \tag{25}$$

Therefore, the probability of reconfiguring when computing n^M_{tasks} is:

$$p_{pr} = \frac{\sum_{j=1}^{n^M_{tasks}} \sum_{i=1}^{n_{types}} p^M_i \cdot (1 - p^M_i)^j}{n^M_{tasks}} \tag{26}$$

The numerator is the value of n_{pr} since it considers all the possible n^M_{tasks}. Notice the difference with Eq. 21. The current strategy searches for a particular task in n^M_{tasks} to avoid reconfiguration while in Eq. 21 there is no search, and therefore, the incoming tasks are randomly selected.

Table 2. *CT of the NE proposed in [4]. Each task emulates one node's configuration, which is determined by the number of active microphones. The compatibility shows that only the same types of tasks are merged. The t_i values are expressed in seconds.*

Task (T_i)	Probability (p_i)	Time Cost (t_i)	Area Cost (a_i)	Compatibility (RM_i)
52 Mics	1/4	1.0834 ± 0.0029	1	RM_{52Mics}
28 Mics	1/4	1.0753 ± 0.0024	1/2	RM_{28Mics}
12 Mics	1/4	1.0679 ± 0.0023	1/4	RM_{12Mics}
4 Mics	1/4	1.0677 ± 0.0023	1/4	RM_{4Mics}

5 Case Study

Our approach is evaluated on the audio streaming application detailed in [4]. This case study is a FPGA-based microphone array network emulator (NE) which has to combine the data received from multiple nodes processing streams of audio. A node of the network supports different configurations based on the number of active microphones, which directly determines the accuracy and the power consumption [3]. The response of the nodes is combined to estimate the location of sound sources, which is used to adapt the networks' topology and the node's configuration to balance the network's power consumption and its accuracy. The computation is repeated an undetermined number of times to evaluate different topologies, sound-sources profiles, node's configurations or data fusion techniques. As a result, tens to hundreds of nodes with different configurations must be evaluated before converging to a valid network configuration. This audio streaming application satisfies the constraints detailed in Section 3:

Non-deterministic scheduling: One execution of the NE demands an unpredictable number of nodes, each with a variable configuration.
Non-priority tasks: The NE needs to collect the node's results without any particular priority order.
Non-data dependencies between tasks: Each node can be considered like one independent task without data dependencies.

5.1 Validation

Our probabilistic approach is validated through experimental simulations using the heuristics proposed in [4]. The achievable speedup is estimated based on the heuristics and their combination, providing an early performance estimation and facilitating the generalized use of PR to accelerate similar applications.

The description of the application in [4] provides enough information to fetch our approach. The evaluation presented here goes, in fact, beyond the original evaluation and multiple t_{pr} and LPs are used to better understand the performance cost and acceleration when using PR. Table 2 is the CT obtained from the node's characteristics. The proposed probabilistic approach is used by considering a task as an emulation of one node of the NE. Therefore, the nodes of the NE are hereby called tasks.

Despite the evaluation of our approach uses the basis of the application, it presents some differences when compared to the experiments done in [4]:

- *Single RP*: The system analysed in [4] considered 4 RPs. For the sake of simplicity, our evaluation only considers one RP ($n_{RP} = 1$). Notice, however, that our equations are general enough to be applied for any n_{RP}.
- *Classification heuristic*: The tasks are not sorted per type during the classification heuristic performed before merging. This initial ordering already improves the followed heuristics and masks their performance contribution, justifying its absence in our approach. Furthermore, it provides a more general evaluation by respecting the original order of the task's execution. It can, nevertheless, be inserted in our equations as additional parameter, but its analysis is out of scope of this paper.
- *Merging heuristic*: The CT shown in Table 2 only considers the merging of the same type of tasks. Therefore, each RM only allocates the same type of task, which is not like in [4]. We consider that it is enough to evaluate the accuracy of our performance prediction.

The impact of the PR is evaluated beyond the configurations detailed in Table 2. The evaluation presented here explores the performance variance based on t_{pr} and A_{cost} to show how the acceleration changes based on the tasks' characteristics detailed in Table 2:

- t_{pr}: The original value of t_{pr} of the system in [4] slightly changes per RP. The average value of t_{pr} is used as reference and scaled by a factor to evaluate adverse situations where t_{pr} is significantly higher than any t_i. The considered scaling factor ranges from 0 to 2. Notice that there is no performance degradation when $t_{pr} = 0$.
- A_{cost}: The original a_i of the tasks is modified to cover a range of A_{cost} (Eq. 4). While originally $A_{cost} = 0.5$, the explored range of A_{cost} varies from 0.3 to 0.875.

Finally, notice that Table 2 provides information about p_i which is not originally available in [4]. Despite there are also 4 types of tasks, each type of task is assumed to be equally probable, since the authors in [4] do not specify this parameter.

5.2 Results

The heuristics introduced in [4] have been implemented and simulated in Matlab 2016b. The tasks' occurrence is expressed through probabilities due to the non-deterministic scheduling. Therefore, the experimental results are average values obtained after 100 executions of 100 tasks. This relatively large number of executions guarantees that the tasks's occurrence is properly represented. Finally, our probabilistic approach is used to predict the theoretical speedup and compared to the experimental one. Notice that both speedups are averaged values due to the non-deterministic nature of the task's execution.

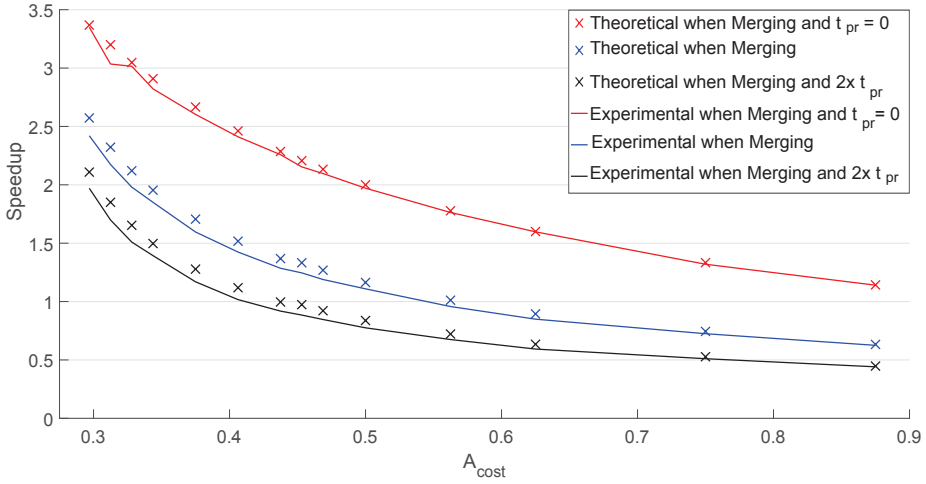

Fig. 3. *Average speedup due to merging tasks. No task's scheduling is applied after merging. Notice the impact of t_{pr} in the performance degradation (Eq. 16).*

Figure 3 depicts the speedup based on A_{cost} when only merging the same type of tasks. The theoretical speedup is obtained through Eq. 12. The experimental speedup without PR cost is obtained when forcing t_{pr} to zero in Eq. 16, leading to no degradation in performance. A scaling factor of one (blue line) and two (black line) are applied to t_{pr}. The highest cost of PR occurs when $t_{pr} = 2 \times t_i$, as shown in the bottom line in Figure 3. The difference between the theoretical speedup and the speedup including the PR cost represents the performance degradation due to PR without any scheduling strategy. Notice that A_{cost} decreases when more tasks can be merged, since their area demanding is lower, leading to an increment of achievable speedup.

Figure 4 shows how the performance increments when applying the iteration-oriented scheduling heuristic described in [4]. This heuristic schedules the tasks based on the RP's previous configuration to reduce the overall n_{pr}. The achievable acceleration is very close to the theoretical acceleration upper bound. The theoretical values, obtained by applying the equations described in Section 4 follow the experimental trend. Nevertheless, this heuristic does not perform as good when n_{RP} increases, as the results in [4] reflect. Their results show a lower performance besides 4 RPs are used. Further analysis on how n_{RP} affects to the achievable acceleration is needed to properly determine the reason of this performance degradation. Different heuristics should be proposed to target multiple RPs in order to achieve a closer performance to the theoretical acceleration upper bound.

The proposed probabilistic model needs to not only characterize the non-deterministic nature of the task's execution but also to reflect the behaviour of the merging and scheduling heuristics required by the general methodology. The comparison between the predicted acceleration and the experimental results

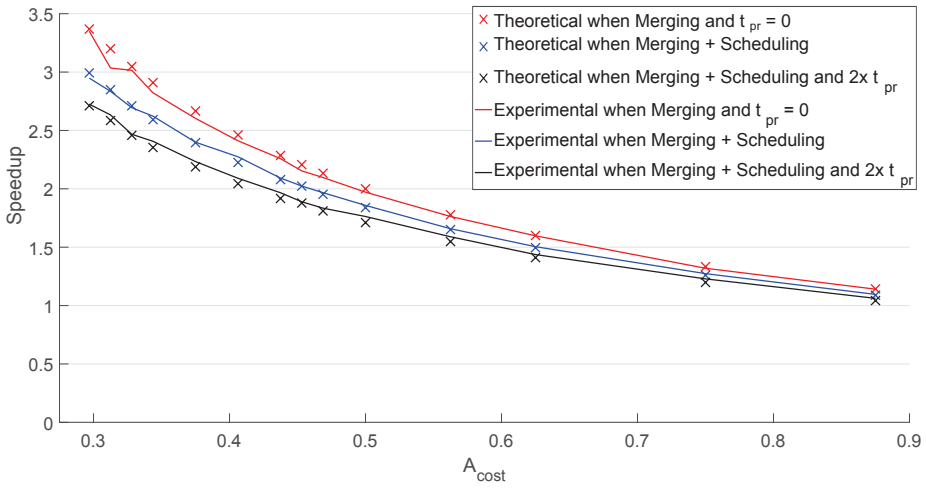

Fig. 4. *Average speedup due to merging and scheduling tasks. The iteration-oriented scheduling is applied after merging.*

depicted in Figure 3 and in Figure 4 demonstrate the accuracy of this model. Nevertheless, the proposed methodology and its probabilistic model are flexible enough to be adapted for different heuristics, like, for instance, the scheduling heuristics proposed in [1].

6 Conclusions

The proposed methodology enables the acceleration of streaming applications with non-deterministic task scheduling using PR. Moreover, the acceleration upper bounds can be predicted at the design time based on the application's characteristics. We believe that many streaming applications can benefit from our approach, specially the ones related to signal processing, to image processing or even to data stream management systems which present a high parallelism and multiple similar configurations. Future work includes the validation of our probabilistic approach for multiple RPs, more case studies and the development of optimized heuristics.

References

1. Cordone, Roberto, et al. *"Partitioning and scheduling of task graphs on partially dynamically reconfigurable FPGAs."* IEEE transactions on computer-aided design of integrated circuits and systems, 2009.
2. da Silva, Bruno, et al. *"Runtime reconfigurable beamforming architecture for real-time sound-source localization."* Field Programmable Logic and Applications (FPL), 26th International Conference on. IEEE, 2016.

3. da Silva, Bruno, et al. *"A partial reconfiguration based microphone array network emulator."* Field Programmable Logic and Applications (FPL), 27th International Conference on. IEEE, 2017.
4. da Silva, Bruno, et al. *"Exploiting Partial Reconfiguration through PCIe for a Microphone Array Network Emulator."* Int. J. Reconfigurable Computing, Hindawi, 2018.
5. da Silva, Bruno, et al. *"A Multimode SoC FPGA-Based Acoustic Camera for Wireless Sensor Networks."* 13th International Symposium on Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC). IEEE, 2018.
6. El-Araby, Esam, et al. *"Performance bounds of partial run-time reconfiguration in high-performance reconfigurable computing."*, In Proceedings of the 1st international workshop on High-performance reconfigurable computing technology and applications: held in conjunction with SC07. ACM, 2007.
7. El-Araby, Esam, et al. *"Exploiting partial runtime reconfiguration for high-performance reconfigurable computing."*, ACM Transactions on Reconfigurable Technology and Systems (TRETS), 2009
8. Gordon, I. Michael et al. *"Exploiting coarse-grained task, data, and pipeline parallelism in stream programs."* ACM SIGARCH Computer Architecture News, 2006.
9. Jimenez, M.I. et al. *"Design of task scheduling process for a multifunction radar."* IET Radar, Sonar & Navigation, 2012.
10. Papadimitriou, Kyprianos, et al. *" Performance of partial reconfiguration in FPGA systems: A survey and a cost model."* ACM Transactions on Reconfigurable Technology and Systems (TRETS), 2011.
11. Malazgirt, Gorker Alp, et al. *"High level synthesis based hardware accelerator design for processing SQL queries."* Proceedings of the 12th FPGAworld Conference. ACM, 2015.
12. Sabatini, Sergio, et al. *"Multifunction array radar- System design and analysis (Book)"*. Norwood, MA: Artech House, 1994.
13. Wildermann, Stefan, et al. *"Self-organizing computer vision for robust object tracking in smart cameras."* In International Conference on Autonomic and Trusted Computing. Springer, 2010.
14. Wildermann, Stefan, et al. *"Placing multimode streaming applications on dynamically partially reconfigurable architectures."* International Journal of Reconfigurable Computing, 2012.
15. Wildermann, Stefan, et al. *"Symbolic system-level design methodology for multimode reconfigurable systems."* Design Automation for Embedded Systems, 2013.